METHOD AND APPARATUS FOR PERFORMING PLAN-BASED DIALOG

RELATED APPLICATIONS

This application is a divisional of U.S. patent application 09/662,242, filed September 14, 2000, and entitled "METHOD AND APPARATUS FOR PERFORMING PLAN-BASED DIALOG".

BACKGROUND OF THE INVENTION

The present invention relates to methods and systems for defining and handling user/computer interactions. In particular, the present invention relates to dialog systems.

Nearly all modern computer interfaces are based on computer driven interactions in which the user must follow an execution flow set by the computer or learn one or more commands exposed by the computer. In other words, most computer interfaces do not adapt to the manner in which the user wishes to interact with the computer, but instead force the user to interact through a specific set of interfaces.

New research, however, has focused on the idea of having a computer/user interface that is based on a dialog metaphor in which both the user and the computer system can lead or follow the dialog. Under this metaphor, the user can provide an initial question or command and the computer system can then identify ambiguity in the question or command and ask refining questions to identify a proper course of action. Note that during the refinement, the user is free to change the dialog and lead it into a new direction. Thus, the computer system must be adaptive

and react to these changes in the dialog. The system must be able to recognize the information that the user has provided to the system and derive a user intention from that information. In addition, the systems must be able to convert the user intention into an appropriate action, such as asking a follow-up question or sending an e-mail message.

Note that the selection of the proper action is critical in that the quality of the user experience is dictated in large part by the number of questions that the system asks the user and, consequently, the amount of time it takes for the user to reach their goal.

In the past, such dialog systems have been created through a combination of technologies. Typically a stochastic model would be used to identify what the user has said. Such models provide probabilities for each of a set of hypothesis phrases. The hypothesis with the highest probability is then selected as the most likely phrase spoken by the user.

This most likely phrase is provided to a natural language parsing algorithm, which applies a set of natural language rules to identify the syntactic and semantic structure of the identified phrase.

The semantic structure is then passed to a plan based system, that applies a different set of rules based on the semantic meaning and the past dialog statements made by the user and the computer. Based on the execution of these rules, the dialog system selects an action that is to be taken.

Some systems have attempted to use stochastic models in the conversion from what was said

to the semantic meaning of what was said. For example, in "The Thoughtful Elephant: Strategies for Spoken Dialog Systems" E. Souvignier et al., IEEE Transactions on Speech and Audio Processing, Vol. 8,

5 No. 1 (January 2000), a stochastic model is applied to both the step of identifying of what has been said and the step of converting what has been said into a semantic meaning.

Other systems have used stochastic models to

10 determine what action to take given a semantic meaning. For example, in "A Stochastic Model for Machine Interaction for Learning Dialog Strategies", Levin et al., IEEE Transactions on Speech and Audio Processing, Vol. 8, No. 1, pg. 11-23 (January 2000), a

15 stochastic model is used in the conversion from a semantic meaning to an action.

Although stochastic models have been used in each of the stages separately, no system has been provided to use stochastic models in all of the stages

20 of a dialog system that are designed to optimize the same objective function. Because of this, the sub-systems in these dialog systems do not integrate naturally with each other.

Another problem with current dialog systems

25 is that they are not well suited for distributed computing environments with less than perfect quality of service. Telephone based dialog systems, for example, rely heavily on the telephone links. A severance in the phone connection generally leads to

30 the loss of dialog context and interaction contents. As a result, the dialog technologies developed for phone based system cannot be applied directly to Internet environments where the interlocutors do not

always maintain a sustained connection. In addition, existing dialog systems typically force the user into a fixed interface on a single device that limits the way in which the user may drive the dialog. For

5 example, current dialog systems typically require the user to use an Internet browser or a telephone, and do not allow a user to switch dynamically to a phone interface or a hand-held interface, or vice versa, in the middle of the interaction. As such, these systems

10 do not provide as much user control as would be desired.

SUMMARY OF THE INVENTION

The present invention provides a dialog system in which the subsystems are integrated under a

15 single technology model. In particular, each of the subsystems uses stochastic modeling to identify a probability for its respective output. The combined probabilities identify a most probable action to be taken by the dialog system given the latest input from

20 the user and the past dialog states.

Specifically, a recognition engine is provided that uses a language model to identify a probability of a surface semantic structure given an input from a user. A semantic engine is also provided

25 that uses a semantic model to identify a probability of a discourse structure given the probability of the surface semantic structures. Lastly, a rendering engine is provided that uses a behavior model to determine the lowest cost action that should be taken

30 given the probabilities associated with one or more discourse structures provided by the semantic engine. By using stochastic modeling in each of the subsystems and forcing all the stages to jointly

optimize a single objective function, the present invention provides a better integrated dialog system that theoretically should be easier to optimize.

5   An additional aspect of the present invention is an embodiment in which the recognition engine, the semantic engine and the rendering engine communicate with one another through XML pages, thus allowing the engines to be distributed across a network. By using XML, the dialog systems can take
10   advantage of the massive infrastructure developed for the Internet.

In this embodiment, the behavior model is written or dynamically synthesized using the extensible stylesheet language (XSL) which allows the
15   behavior model to convert the XML pages generated by the semantic engine into an output that is not only the lowest cost action given the discourse representation found in the semantic engine XML page, but is also appropriate for the output interface
20   selected by the user. In particular, the XSL-transformations provided by the behavior model allow a single XML page output by the semantic engine to be converted into a format appropriate for an Internet browser, a phone system, or a hand-held system, for
25   example. Thus, under this embodiment, the user is able to control which interface they use to perform the dialog, and in fact can dynamically change their interface during the dialog.

BRIEF DESCRIPTION OF THE DRAWINGS

30   Fig. 1 is a general block diagram of a personal computing system in which the present invention may be practiced.

Fig. 2 is a block diagram of a dialog system of the present invention.

Fig. 3 is a flow diagram for a dialog method under the present invention.

Fig. 4 is a graphical representation of a surface semantic tree structure generated by a recognition engine for a speech input.

Fig. 5 is graphical representation of a surface semantic structure created by a recognition engine for a pointer-device input.

Fig. 6 is a graphical representation of a discourse tree structure generated by a discourse engine of the present invention.

Fig. 7 is the discourse tree of Fig. 6 showing the message node after it has been collapsed.

Fig. 8 is the discourse tree of Fig. 7 showing an expansion of the discourse tree to include meeting entries found in a domain table.

Fig. 9 is a graphical representation of a surface semantic structure generated by the recognition engine based on the user's response to a system question.

Fig. 10 is the discourse tree of Fig. 8 after the existing meeting node has been collapsed in response to the user's answer.

Fig. 11 is the discourse tree of Fig. 10 after the meeting attendees node has been collapsed.

Fig. 12 is the discourse tree of Fig. 11 after the recipients node has been collapsed.

Fig. 13 is a block diagram of a second embodiment of the dialog system of the present invention.

Fig. 14 is a flow diagram of a mark-up language based embodiment of a dialog system under the present invention.

DETAILED DESCRIPTION OF ILLUSTRATIVE EMBODIMENTS

FIG. 1 illustrates an example of a suitable computing system environment 100 on which the invention may be implemented. The computing system environment 100 is only one example of a suitable computing environment and is not intended to suggest any limitation as to the scope of use or functionality of the invention. Neither should the computing environment 100 be interpreted as having any dependency or requirement relating to any one or combination of components illustrated in the exemplary operating environment 100.

The invention is operational with numerous other general purpose or special purpose computing system environments or configurations. Examples of well-known computing systems, environments, and/or configurations that may be suitable for use with the invention include, but are not limited to, personal computers, server computers, hand-held or laptop devices, multiprocessor systems, microprocessor-based systems, set top boxes, programmable consumer electronics, network PCs, minicomputers, mainframe computers, telephony systems, distributed computing environments that include any of the above systems or devices, and the like.

The invention may be described in the general context of computer-executable instructions, such as program modules, being executed by a computer. Generally, program modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. The invention may also be

practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote computer storage media including memory storage devices.

With reference to FIG. 1, an exemplary system for implementing the invention includes a general purpose computing device in the form of a computer 110. Components of computer 110 may include, but are not limited to, a processing unit 120, a system memory 130, and a system bus 121 that couples various system components including the system memory to the processing unit 120. The system bus 121 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. By way of example, and not limitation, such architectures include Industry Standard Architecture (ISA) bus, Micro Channel Architecture (MCA) bus, Enhanced ISA (EISA) bus, Video Electronics Standards Association (VESA) local bus, and Peripheral Component Interconnect (PCI) bus also known as Mezzanine bus.

Computer 110 typically includes a variety of computer readable media. Computer readable media can be any available media that can be accessed by computer 110 and includes both volatile and nonvolatile media, removable and non-removable media. By way of example, and not limitation, computer readable media may comprise computer storage media and communication media. Computer storage media includes both volatile and nonvolatile, removable and non-

removable media implemented in any method or technology for storage of information such as computer readable instructions, data structures, program modules or other data. Computer storage media

5 includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical disk storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any

10 other medium which can be used to store the desired information and which can be accessed by computer 100.

Communication media typically embodies computer readable instructions, data structures, program modules or other data in a modulated data signal such

15 as a carrier wave or other transport mechanism and includes any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way

20 of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, FR, infrared and other wireless media. Combinations of any of the above should also be

25 included within the scope of computer readable media.

The system memory 130 includes computer storage media in the form of volatile and/or nonvolatile memory such as read only memory (ROM) 131 and random access memory (RAM) 132. A basic

30 input/output system 133 (BIOS), containing the basic routines that help to transfer information between elements within computer 110, such as during start-up, is typically stored in ROM 131. RAM 132 typically

contains data and/or program modules that are immediately accessible to and/or presently being operated on by processing unit 120. By way o example, and not limitation, FIG. 1 illustrates operating system 134, application programs 135, other program modules 136, and program data 137.

The computer 110 may also include other removable/non-removable volatile/nonvolatile computer storage media. By way of example only, FIG. 1 illustrates a hard disk drive 141 that reads from or writes to non-removable, nonvolatile magnetic media, a magnetic disk drive 151 that reads from or writes to a removable, nonvolatile magnetic disk 152, and an optical disk drive 155 that reads from or writes to a removable, nonvolatile optical disk 156 such as a CD ROM or other optical media. Other removable/non-removable, volatile/nonvolatile computer storage media that can be used in the exemplary operating environment include, but are not limited to, magnetic tape cassettes, flash memory cards, digital versatile disks, digital video tape, solid state RAM, solid state ROM, and the like. The hard disk drive 141 is typically connected to the system bus 121 through a non-removable memory interface such as interface 140, and magnetic disk drive 151 and optical disk drive 155 are typically connected to the system bus 121 by a removable memory interface, such as interface 150.

The drives and their associated computer storage media discussed above and illustrated in FIG. 1, provide storage of computer readable instructions, data structures, program modules and other data for the computer 110. In FIG. 1, for example, hard disk drive 141 is illustrated as storing operating system

144, application programs 145, other program modules 146, and program data 147. Note that these components can either be the same as or different from operating system 134, application programs 135, other program

5   modules 136, and program data 137. Operating system 144, application programs 145, other program modules 146, and program data 147 are given different numbers here to illustrate that, at a minimum, they are different copies.

10   A user may enter commands and information into the computer 110 through input devices such as a keyboard 162, a microphone 163, and a pointing device 161, such as a mouse, trackball or touch pad. Other input devices (not shown) may include a joystick, game

15   pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit 120 through a user input interface 160 that is coupled to the system bus, but may be connected by other interface and bus structures, such

20   as a parallel port, game port or a universal serial bus (USB). A monitor 191 or other type of display device is also connected to the system bus 121 via an interface, such as a video interface 190. In addition to the monitor, computers may also include other

25   peripheral output devices such as speakers 197 and printer 196, which may be connected through an output peripheral interface 190.

        The computer 110 may operate in a networked environment using logical connections to one or more

30   remote computers, such as a remote computer 180. The remote computer 180 may be a personal computer, a hand-held device, a server, a router, a network PC, a peer device or other common network node, and

typically includes many or all of the elements described above relative to the computer 110. The logical connections depicted in FIG. 1 include a local area network (LAN) 171 and a wide area network (WAN)

5   173, but may also include other networks. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the Internet.

When used in a LAN networking environment,

10 the computer 110 is connected to the LAN 171 through a network interface or adapter 170. When used in a WAN networking environment, the computer 110 typically includes a modem 172 or other means for establishing communications over the WAN 173, such as the Internet.

15 The modem 172, which may be internal or external, may be connected to the system bus 121 via the user input interface 160, or other appropriate mechanism. In a networked environment, program modules depicted relative to the computer 110, or portions thereof, may

20 be stored in the remote memory storage device. By way of example, and not limitation, FIG. 1 illustrates remote application programs 185 as residing on remote computer 180. It will be appreciated that the network connections shown are exemplary and other means of

25 establishing a communications link between the computers may be used.

Fig. 2 provides a block diagram of a dialog system of the present invention. FIG. 2 is described below in connection with a dialog method shown in the

30 flow diagram of FIG. 3.

Under one embodiment of the invention, the components of Fig. 2 are located within a personal computer system, such as the one shown in Fig. 1. In

other embodiments, the components are distributed across a distributed computing environment and connected together through network connections and protocols. For example, the components could be
5 distributed across an intranet or the Internet. A specific embodiment of the dialog system of the present invention designed for such a distributed computing environment is discussed in more detail below with reference to the block diagram of Fig. 13.
10 In Fig. 2, the dialog system 200 receives input from the user through a plurality of user interfaces 202, 204. Such user input interfaces include a speech capture interface capable of converting user speech into digital values, a
15 keyboard, and a pointing device, such as a mouse or track ball. The present invention is not limited to these particular user input interfaces and additional or alternative user input interfaces may be used with the present invention.
20 Each user input interface is provided to its own recognition engine 206, 208 which has an associated language model 210, 212. Recognition engines 206 and 208 use language models 210 and 212, respectively, to identify and score possible surface
25 semantic structures to represent the respective inputs. Each recognition engine 206, 208 provides at least one surface semantic output and a score representing the probability of that semantic output. In some embodiments, at least one of the recognition
30 engines 206, 208 is capable of providing more than one alternative surface semantic structure with an associated score for each alternative structure. Each of the semantic structures and corresponding scores is

provided to discourse engine 214. The step of generating the surface semantics is shown as step 300 in Fig. 3.

For language-based user input such as speech
5 and handwriting, the language model used by the recognition engine can be any one of a collection of known stochastic models. For example the language model can be an N-gram model that models the probability of a word in a language given a group of N
10 proceeding words in the input. The language model can also be a context free grammar that associates semantic and and/or syntactic information with particular words and phrases. In one embodiment of the present invention, a unified language model is
15 used that combines an N-gram language model with a context free grammar. In this unified model, semantic and/or syntactic tokens are treated as place values for words and an N-gram probability is calculated for each hypothesized combination of words and tokens.

20 In several embodiments, the language model is capable of generating a hierarchical surface semantic structure that is similar to a discourse semantic structure defined in a discourse model 216 and used by a discourse engine 214. By using similar
25 hierarchical structures in both models, it becomes easier to translate the recognized input values from the surface semantic structure to the discourse semantic structure. Note that in many embodiments, language models associated with non-linguistic inputs
30 such as the pointing device, are also capable of attaching semantic tokens to the pointing device input. In most embodiments, the semantic tokens are taken from a set of semantic tokens found in the

discourse semantic structure. Thus, when a user clicks on a file icon with the mouse, the recognition engine for the mouse is capable of associating a FileName token with that act while pointing to the ID

5 of the indicated file as the input.

As shown in step 302 of Fig. 3, when discourse engine 214 receives the surface semantics from recognition engines 206, 208, it first expands a discourse semantic tree. This expansion is performed

10 first by examining the surface semantic structures provided to discourse engine 214. If the surface semantic structures provided to discourse engine 214 show a previously unseen high level semantic token, i.e., a digression in the dialog, a new discourse tree

15 is instantiated in discourse engine 214.

If, on the other hand, the surface semantics provided to discourse engine 214 contain low level discourse tokens, discourse engine 214 first looks to see if the surface semantic tokens can fit in any

20 currently opened discourse tree. If the semantic tokens can fit in a currently opened discourse tree, the tokens and their associated input values are placed in the appropriate slots of the discourse semantic structure.

25 If the semantic tokens do not fit within any currently active discourse structures, the discourse engine 214 looks for possible discourse trees in which the semantic tokens can be found. If a surface semantic token can be inserted into more than one

30 structure or into more than one location within a structure, a situation known as semantic ambiguity, discourse engine 214 expands all the structures in which the token could be inserted, but does not insert

the semantic token. Instead, discourse engine 214 holds the semantic token and its associated input value in a discourse memory 218 until the ambiguity as to where the semantic object should be placed has been

5 resolved. Typically, this ambiguity is resolved by requesting additional information from the user, as discussed further below.

In most embodiments, the recognition engine is able to associate a surface semantic token with one

10 or more input words even though the words themselves are ambiguous. For example, in a phrase such as "send it to him" the recognition engine may be able to identify that "it" refers to a message object and "him" refers to a recipient. Thus, the recognition

15 engine is able to associate a Message token with "it" and a Recipient token with "him". However, it is not clear from the words "it" and "him" which message should be sent or which person should receive the message.

20 Under the present invention, discourse engine 214 attempts to clarify such ambiguities by using a discourse memory 218, which contains past values for specific semantic token types. By referring to this memory of past references to

25 semantic token types, the system of the present invention is able to infer values for implicit references based on past discourse statements. Thus, using the example above, if a particular message had appeared recently in the discussion and a particular

30 person had been referred to in the discussion, discourse engine 214 would replace the phrase "it" with the particular message ID and "him" with the particular person's name.

Under one embodiment of the present invention, this inferential ability is facilitated by organizing discourse memory 218 into separate priority queues. In particular, under this embodiment of the invention, discourse memory 218 is divided into a short-term memory containing values from the current user input and a long-term memory containing values from past user input.

The short-term memory is further divided into an explicit memory and an implicit memory. The explicit memory holds values that have been resolved directly from input provided by the user. For instance, if the user refers to a person by name in the beginning of a sentence, the name is placed in the explicit memory under the Person token type. The implicit memory holds values that have been resolved from indirect references made by the user such as anaphora (where an item takes its meaning from a preceding word or phrase), ellipsis (where an item is missing but can be naturally inferred), and deixis (where an item is identified by using definite articles or pronouns). Examples of such implicit references include statements such as "Send it to Jack", where "it" is an anaphora that can be resolved by looking for earlier references to items that can be sent or "Send the message to John's manager" where "John's manager is a deixis that is resolved by searching through a database, described later, to find who John's manager is. This name is then placed in the implicit memory for later use.

Under one embodiment, the three memories are prioritized such that the system looks in the explicit memory first, the long-term memory second, and the

implicit memory last when attempting to resolve a reference. Thus, if a person has been explicitly referred to in the current input from the user, that person's name would take priority over a person's name that was found in the long-term memory, or a person's name that had been resolved and placed in the implicit memory.

Under one embodiment, the priority ordering of discourse memory 218 is ignored if a higher priority memory value is inconsistent or contradicts other input provided by the user. For example, if the user refers to "her" but the last name in the explicit memory is a male name, the priority of the explicit memory would be ignored in favor of looking for the first female name in the long-term memory or the implicit memory.

Under one embodiment, before expanding the discourse semantic structure, discourse engine 214 updates discourse memory 218 while resolving indirect references in the current input from the user. In particular, discourse engine 214 updates the explicit memory based on any explicit terms found in the current input and updates the implicit memory by resolving indirect references in the current input. Under this embodiment, the resolution is done on a first-in-last-out basis for the input from the user such that explicit and implicit values in the first part of the user input are used to resolve indirect references found in later parts of the user input.

After implicit references have been resolved using discourse memory 218, the values retrieved from memory 218 are associated with their respective

semantic token in the expanded discourse semantic structure.

Once discourse engine 214 has expanded the discourse semantic structure, it attempts to collapse the discourse semantic structure as much as possible at step 304. To collapse the discourse semantic structure, discourse engine 214 looks at each bottom level token to determines whether it has enough information about the semantic token to identify a single entity that can replace the token. In this context, an entity is an entry in one of a set of domain tables 220 that are accessed by one or more domain experts 222. The domain experts 222 identify which table needs to be accessed and updated, and handle the overhead and protocols associated with accessing the tables.

For example, to collapse a Person semantic token, which is a general representation of a person, discourse engine 214 attempts to find a single person that meets the attributes currently associated with the Person token. To do this, discourse engine 214 passes the attributes and the token to a domain expert 222 specializing at resolving people, which then accesses a plurality of domain tables that are associated with the Person token. In this case, domain experts 222 would access a domain table that contains a list of people.

In the person domain table, each row is identified as an entity, i.e. a person, and each column is an attribute of the person. Thus, determining whether a token can be collapsed involves determining whether the attributes of the token provide enough information that a single entity can be

identified in the domain tables 220. If a single entity can be found in domain tables 220, the entity is inserted in place of the token in the discourse semantic structure.

In an embodiment that uses a tree structure for the discourse semantic structure, the semantic tokens appear as nodes on the tree, and the attributes of the token appear as children of the node. In such embodiments, discourse engine 214 attempts to collapse the discourse tree from the bottom up so that children nodes are collapsed first and the resolution of the nodes "bubbles up".

If a token can not be collapsed because more than one entity in the domain tables 220 meet the available search criteria, discourse engine 214 may choose to retrieve all of the matching entities up to some maximum number of entities. Discourse Engine 214 may also utilize the discourse model 216 to aid the evaluation process in step 304 and discard hypotheses with low probabilities. Either way, discourse engine 214 then augments the discourse semantic structure with these alternative possibilities extending from the unresolved token.

At step 306 of FIG. 3, discourse engine 214 uses discourse model 216 to generate scores that describe the likelihood of each discourse semantic structure. In this context, each entity extending from an unresolved semantic token is considered to be part of a separate discourse semantic structure, even though they share common semantic tokens. Thus, discourse engine 214 generates a separate score for each entity extending from an unresolved token. Note that in some embodiments, step 304 and step 306 can be

effectively combined into a single step for performance considerations.

As an example, if the user had stated that they wanted to "Send an e-mail to John", but domain tables 220 contained a John A, a John B, and a John C, discourse engine 214 would generate separate scores for sending e-mail to John A, John B, and John C. If in the past, the user has sent an equal number of e-mails to John A, John B, and John C, the scores generated by discourse engine 214 and discourse model 216 would be equal for each semantic structure. However, if the user sends e-mail to John A 90% of the time, John B 8% of the time, and John C 2% of the time, the scores generated by discourse model 216 will be heavily weighted toward John A and will be quite low for John B and John C.

In many embodiments, the same mechanisms for resolving semantic ambiguity may be used to resolve recognition ambiguity. For example, when a user says "Send e-mail to John A," a speech recognizer may recognize the utterance as to "John A", "John K", or "John J" based on acoustic and other confounding factors. In many embodiments, the system may choose to view these competing recognition hypotheses as a semantic ambiguity. This treatment eliminates extra handling that would be needed if the recognition ambiguity was resolved before the surface semantic was given to the discourse engine.

The ambiguity resolution can also be extended to semantic contradictions that arise in a multimodal environment in which multiple recognition engines each provide surface semantics that contradict one another. For example, a user says "Send e-mail to

John A" but clicks on the picture of "John B" on the display. In many embodiments, the cross modality semantic ambiguity can be treated in the same manner described above without executing special instructions
5   to handle cross-modality conflicts.

At step 308, rendering engine 224 of Fig. 2 receives the discourse semantic structure generated by discourse engine 214 as well as the scores associated with each path through the structure. Rendering
10   engine 224 uses the discourse semantic structure and the associated scores as input to a behavior model 226 which generates the cost of taking particular actions in view of the available user interfaces and the current state of the dialog as represented by the
15   discourse semantic structure.

The cost of different actions can be calculated based on several different factors. For example, since the usability of a dialog system is based in part on the number of questions asked of the
20   user, one cost associated with a dialog strategy is the number of questions that it will ask. Thus, an action that involves asking a series of questions has a higher cost than an action that asks a single question.

25   A second cost associated with dialog strategies is the likelihood that the user will not respond properly to the question posed to them. This can occur if the user is asked for too much information in a single question or is asked a
30   question that is too broadly worded.

Lastly, the action must be appropriate for available output user interface. Thus, an action that would provide multiple selections to the user would

have a high cost when the output interface is a phone because the user must memorize the options when they are presented but would have a low cost when the output interface is a browser because the user can see all of the options at once and refer to them several times before making a selection.

Under the embodiment of Fig. 2, discourse engine 214 provides the available user interfaces to rendering engine 224 from an interface memory 230. Note that in other embodiments, interface memory 230 may be connected directly to rendering engine 224, or rendering engine 224 may access an operating system function to identify the available output interfaces.

When determining the cost of an action, rendering engine 224 and behavior model 226 will consider whether a semantic structure has a high enough score that the rendering engine has a high likelihood of success simply by performing the action associated with the discourse semantic structure. For example, if the user had said "Send this message to John", as in the example noted above, and the score for John A was significantly higher than the scores for John B and John C, the rendering engine will simply send the message to John A without asking for further clarification from the user. In such an instance, the cost of making an error in sending the e-mail to John A would be less than the cost of asking the user to clarify which John they wish to send the e-mail message to. On the other hand, if the cost of making an error in sending the e-mail to John A is high, the proper action would be to generate a confirmation query to the user. In many embodiments,

the costs and thresholds can be expressed explicitly in the behavior model 226.

In step 310 of Fig. 3, rending engine 224 selects the highest action score and executes the corresponding action. Often times, this involves sending a response to the user through a user output interface 228.

After rendering engine 224 has selected an action, in some embodiments, it modifies one or more of the language models 210 and 212 so that the language models can be used to properly interpret the user's response to the action. For example, if the rendering engine supplies three alternatives to the user, it can update the language model to associate a phrase such as "the first one", or "the second one" with particular entities. This allows the recognition engine to replace the identified phrases with the particular entities, making it easier for the discourse engine 214 to insert the entities in the proper slot of the discourse semantic structure.

As a further explanation of the operation of the discourse system described in Figs. 2 and 3, Figs. 4-12 provide examples of surface semantic structures, and discourse structures for a sample dialog. In the discussion below, tree structures are used for the surface semantic structure and the discourse structure. As noted above, in such structures, the "children" of a node in the tree can be viewed as an attribute of the token at that node. In the discussion below, child nodes are referred to as children and attributes interchangeably. In addition, for the purposes of the discussion below, the recognition engines are assumed to be well-behaved and

do not generate results that cause semantic or recognition ambiguities even though these situations can be handled properly in this invention.

The discourse begins with the user saying "Send it to those in the meeting on Wednesday". This phrase is received by a speech recognition engine, which uses a language model to generate the surface semantic structure shown in Fig. 4. The surface semantic structure of Fig. 4 includes a root node 400 containing the semantic token <Send mail>, which is associated with the entire phrase, "Send it to those in the meeting on Wednesday". The <Send mail> token has two children nodes, or attributes, 402 and 404 which are identified as semantic tokens <Message> and <Recipient>, respectively. <Message> token 402 is associated with the word "it" and <Recipient> token 404 is associated with the phrase "those in the meeting on Wednesday".

<Recipient> token 404 has a further child node 406 that contains the semantic token <Meeting attendees>, which is associated with the phrase "those in the meeting on Wednesday".

<Meeting attendees> token 406 has an attribute of <Existing meeting>, which is represented by token 408 and is associated with the phrase "meeting on Wednesday". <Existing meeting> token 408 has an attribute 410 that contains the semantic token <Date>, which is associated with the word "Wednesday".

Note that the recognition engine associated with the semantic tree structure of Fig. 4 is only one recognition engine that may be operating on the system. Fig. 5 shows a surface semantic structure produced by a recognition engine operating in parallel

with the speech recognition engine. In particular, the surface semantic structure of Fig. 5 is generated by a recognition engine associated with a pointing device. In the present example, the pointing device generates a signal indicating that the user clicked on a particular message while they were saying "Send it to those in the meeting on Wednesday". Based on this clicking gesture, the recognition engine associated with the pointing device generates a single node structure 500 containing a <Message> semantic token that is associated with the message ID of the message clicked on by the user.

The surface semantic structures of Figs. 4 and 5 are provided to the discourse engine which first tries to expand discourse semantic structures by inserting the current input information into as many nodes as possible. In the current example, this results in a discourse semantic tree as shown in Fig. 6.

The discourse semantic tree of Fig. 6 includes a root node 600 having a <Send mail> semantic token. <Send mail> token 600 has six attributes, which appear as children nodes 602, 604, 606, 608, 610, and 612, containing semantic tokens of <Subject>, <Attachment>, <Blind copy>, <Carbon copy>, <Message>, and <Recipient>, respectively.

Based on the surface semantic structures of Figs. 5 and 6, discourse engine 214 has two possible values that can be associated with <Message> token 610. These values are "it" and the message ID returned by the pointing device recognition engine.

<Recipient> node 612 points to further possible recipient types including <Person> token 614,

<Meeting attendees> token 616 and <Work group> token 618. Of these three tokens, only <Meeting attendees> 616 has been further expanded to include a child node of <Existing meeting> 620.

5   <Existing meeting> token 620 represents an existing meeting in a database. Such existing meetings can be referred to by subject, date, the place of the meeting, the organizer of the meeting, or the meeting ID. Each of these attributes is shown as

10 a separate token 622, 624, 626, 628, and 630, respectively. Based on the surface semantics, the discourse engine is able to associate a value of "Wednesday" with <Date> token 624. However, the surface semantics did not provide values for the other

15 attributes of <Existing meeting> token 620.

               Although all the attributes are shown for each semantic token of Fig. 6, even though the attributes are not filled, in other embodiments, these attributes would not be included as nodes for the

20 discourse structure until a surface semantic indicated that they should be added to the larger discourse tree.

               After expanding the discourse semantic tree, the discourse engine then attempts to collapse the

25 tree as much as possible. Using the tree of Fig. 6, discourse engine 214 is able to collapse <Message> token 610 by first resolving the indirect reference "it" into the direct and explicit reference of message ID and then confirming through the domain experts that

30 there is only one message with this message ID. Thus, the generalized <Message> token 610 is replaced by the specific message ID entity, as shown in Fig. 7.

Discourse engine 214 then attempts to collapse the lowest node on the recipient branch of the discourse structure. This involves attempting to collapse <Existing meeting> token 620. To do this, the discourse engine uses a meeting domain expert 222 to search domain tables 220 for an existing meeting that has the attributes attached to <Existing meeting> token 620. In the example of Fig. 6 this involves searching the database for a meeting that occurred on a Wednesday. If there is only one meeting that occurred on a Wednesday, <Existing meeting> token 620 is replaced by the identification number for that meeting.

If, however, there is more than one meeting that occurred on a Wednesday, the domain expert returns all of the meetings that match the search criteria. Discourse engine 214 adds these meetings as possible values for <Meeting ID> token 630. This is shown in Fig. 8 with alternatives 800, 802, and 804 extending from <Meeting ID> token 630.

Since existing meeting 620 cannot be collapsed, discourse engine 214 then uses discourse model 216 to determine a probability that the user wishes to send mail to attendees of each of the three possible meetings 800, 802 and 804. Thus, discourse engine 214 generates three separate scores for the <Send mail> root node.

If, based on the discourse model, none of the structures associated with the possible meetings has a high enough score to make asking a refining question more costly than sending an e-mail, rendering engine 224 asks a question of the user to clarify which of the meetings the user was referring to.

Under one embodiment, rendering engine 224 will also update the language model to accept input such as "the first one" or the "the second one" so that such input can be associated with a particular meeting based on how the rendering engine provides the options to the user.

After the user has been asked the meeting refinement question, the speech recognition engine receives the phrase "the first one" from the user. Based on the modified language model, the recognition engine is able to convert this input into a surface semantic structure having a root token of <Meeting> that is associated with the identification number of the first meeting. Such a surface semantic structure is shown in Fig. 9.

Based on this new surface semantic structure, discourse engine 214 once again attempts to expand the send mail discourse structure. In this case, the information provided is an entity for the <Meeting ID> token, which is thus associated with the <Meeting ID> token.

After this small expansion, discourse engine 214 attempts to collapse as many nodes as possible of the send mail discourse structure. The first node to be collapsed is the <Meeting ID> token. This is done simply by replacing the token with the meeting ID that was associated with it during the expansion.

Next, discourse engine 214 attempts to collapse the <Existing meeting> token. Since the <Meeting ID> attribute of the <Existing meeting> token has been replaced by a meeting ID entity, the <Existing meeting> token can be collapsed by replacing the <Existing meeting> token with the meeting ID.

The next token that can be collapsed is <Meeting Attendees> token 618. To collapse this token, discourse engine 214 passes the meeting ID to the domain experts which search the appropriate domain table to identify the people who attended the meeting associated with that meeting ID. The domain experts then return the identification of each of the people who attended the meeting. The identifications for these people are then put in place of the <Meeting attendees> token as shown in Fig. 11.

Discourse engine 214 then attempts to collapse the <Recipient> token based on the people listed below it. To do this, discourse engine 214 passes the identifications for these people to the domain experts, which search the appropriate domain table to identify whether these people have e-mail addresses. If these people have e-mail addresses, the e-mail addresses are returned by the domain experts. The e-mail addresses are then put in place of the <Recipient> token. This is shown in Fig. 12.

At this stage, discourse engine 214 uses the discourse model to attribute a score to the entire send mail discourse structure. This score will take into account both the recognition score associated with the input used to generate the send mail discourse structure, as well as the likelihood that the user would want to send mail. Note also that the discourse model attributes a higher score to discourse structures that have little ambiguity. In this case, since both the recipients and a message to send have been identified, the send mail discourse structure has a very high probability since it is likely that the user intends to send this message ID to the e-mail

addresses listed in the send mail discourse structure of Fig. 12.

The discourse structure of Fig. 12 and its related probabilities are passed to the rendering engine which uses the behavior model to determine whether the probability is high enough to execute the action represented by the send mail discourse structure. Since the send mail discourse structure is fully collapsed, it is very likely that the rendering engine will send the e-mail.

As described above, each of the subsystems of the present dialog system use stochastic models to perform pattern recognition. Thus, recognition engine 206 uses language model 210 to identify probabilities for various different surface semantic structures. This can be represented mathematically as $P(F|x)$ where x is the user input, F is the identified surface semantic structure, and $P(F|x)$ is the probability of a surface semantic structure given the user input.

Similarly, the discourse engine uses a discourse model to produce scores for each of a set of possible discourse structures. Generically, the discourse engine can be thought of as providing the probability of a current dialog state given a previous dialog state and the surface semantic structure. In terms of a mathematical representation, this would appear as $P(S_n|F, S_{n-1})$ where S_n is the current dialog state, F is the current surface semantic structure, S_{n-1} is the previous dialog state, and $P(S_n|F, S_{n-1})$ is the probability of the current dialog state given the current surface semantic structure and the previous dialog state. Note that in this context, the previous dialog state includes the discourse memory and any

discourse structures that had been opened by the discourse engine.

Thus, taken together, recognition engine 206, language model 210, discourse engine 214 and discourse model 216 represent a dialog state engine that identifies a score for a possible current dialog state based on the user input and a past dialog state. Those skilled in the art will recognize that although the dialog state engine has been described as containing two smaller engines, in other embodiments, the dialog state engine is implemented as a single engine that uses a single model.

Under the present invention, rendering engine 224 also utilizes a stochastic model, represented by behavior model 226. Specifically, rendering engine 224 determines the lowest cost action given the current dialog state. This can be represented as determining the probability of each action given the current dialog state or mathematically as $P(A|S_n)$ where A is an action and S_n is a current dialog state.

Because each element of the dialog system under the present invention uses a stochastic model, the actions of the dialog system can be represented as a single stochastic model. In terms of a mathematical equation, the function of the dialog system can be represented as

$$A_{opt} = \arg\max_A P\left(A|x, S_{n-1}\right) \qquad \text{EQ. 1}$$

where A_{opt} is the optimum action to be taken, and $P(A|x, S_{n-1})$ is the probability of an action, A, given a user input, x, and a previous dialog state, S_{n-1}.

The overall probability given by equation 1 can be broken down further into the individual probabilities associated with the recognition engine, the discourse engine, and the rendering engine. This produces:

$$A_{opt} = \arg\max_A \Sigma_S P(A|S_n) \Sigma_F P(S_n|F, S_{n-1}) P(F|x) \quad \text{EQ. 2}$$

where $P(A|S_n)$ is the probability generated by the rendering engine, $P(S_n|F, S_{n-1})$ is the probability generated by the discourse engine, and $P(F|x)$ is the probability generated by the recognition engine. Using the Viterbi approximation, Equation 2 can be further simplified to:

$$A_{opt} \approx \arg\max_{A,S} P(A|S_n) \Sigma_F P(S_n|F, S_{n-1}) P(F|x) \quad \text{EQ. 3}$$

where the Viterbi approximation selects the largest probability in the rendering engine to represent the sum of the probabilities for all of the possible actions.

The ability to represent the entire system performance in a single mathematical equation makes it easier to optimize the present system. Theoretically and empirically, this has been shown to provide a more efficient means for training the system as a whole and for integrating the various sub-systems to produce the

single system. Furthermore, the separate probabilities associated with each sub-system provide a means for forming the sub-systems into modular units such that the exact function of each modular unit is

5 not important as long as the modular unit gives a score for an expected output structure when given a set of input values.

Fig. 13 provides a block diagram of a second embodiment of the present invention that may be

10 practiced across a distributed computing environment. FIG. 14 provides a flow diagram of a dialog method using the embodiment of FIG. 13.

In Fig. 13, the user provides input through one or more user input interfaces represented by

15 interfaces 1300 and 1302. The user input can be transmitted across networks such as the Internet or an intranet as shown by networks 1304 and 1306. The input may be transmitted according to any one of a variety of known network protocols including HTTP and

20 FTP. In particular, speech input from the user may be sent as a wave file, or as a collection of feature vectors. User input may also be transmitted in an extensible markup language (XML) format.

At step 1400 of FIG. 14, recognition engines

25 1308 and 1310 use language models 1312 and 1314 to identify a most likely set of surface semantics based on the user input. Under this embodiment of the invention, language models 1312 and 1314 are authored in the extensible mark-up language (XML) format.

30 Under this XML format, the different surface semantic objects are represented as tags in the XML page and the hierarchy of the semantic objects is represented by the hierarchy found in the XML nested tags.

The tags from the language model that are associated with the user input by the recognition engine are placed into a separate XML description page that conveys the surface semantics. Under one embodiment, a mark-up language, referred to as a semantic mark-up language or SML, is extended from XML to represent the surface semantics. . Under this embodiment, the output of recognition engines 1308 and 1310 is a semantic mark-up language page. For example, the surface SML for an utterance "what is the phone number for Kuansan?" is:

```
<DirectoryQuery...>
  <PersonByName type="Person" parse="Kuansan">
    kuansan
  </PersonByName>
  <DirectoryItem type="DirectoryItem" parse="phone
    number" confidence="65">
  </DirectoryItem>
</DirectoryQuery>
```

In this example, DirectoryQuery represents the root node of the surface semantics indicating what the basic type of intention found in the utterance. PersonByName indicates that there is a person explicitly referred to in the utterance and DirectoryItem indicates that the user is looking for a directory item.

In many embodiments, the instructions to generate valid SML pages are dynamically synthesized and embedded in the language model. In most embodiments the instructions to generate valid SML pages follow an extensible stylesheet language transformations (XSLT) standard set by the Worldwide Web Consortium. The XSLT standard consists of two integrated portions: matching the source document and generating the transformed document. For user inputs

that are formatted as XML pages, both portions of the standard are used by the language model. However, for user input that is not formatted as XML pages, such as wave files, straight text or feature vectors, only the
5   standard for generating transformed documents can be used.

In the SML page, the recognition scores are tagged to the SML nodes as XML attributes. One example of passing the confidence measure on the DirectoryItem
10   is shown above. In some embodiments, the scores of acoustic models and language models are attached in the same fashion.

The SML pages generated by recognition engines 1308 and 1310 are passed to the discourse
15   engine 1314. Under the embodiment of Fig. 13, discourse engine 1314 may be located on the same machine as the recognition engines or may be located on a remote machine, in which case the SML pages are transmitted through a network, such as network 1350 of
20   FIG. 13, to the discourse engine.

At step 1402 of FIG. 14, discourse engine 1314 uses a discourse model 1316 to convert the surface semantic SML page into a discourse semantic SML page. Under one embodiment of the invention, this
25   is done using a specialized XML to specify the discourse model. In particular, the discourse model is written in a semantic definition language (SDL) which defines the legitimate constructs of a SML document and specifies the relationships among the
30   semantic objects found in the surface semantics and the discourse semantics. Using SDL to define the mark-up language schema for SML allows the system to dynamically adjust the schema for SML, and eliminates

the need for a separate SML schema specification in either the Document Type Definition or the XML Schema format. The SDL pages of the model also provide semantic inference rules that discourse engine 1314

5 utilizes to expand and contract the discourse structures. This includes the rules for accessing the discourse memory 1318, and the domain experts 1320. Note that under this invention, the discourse memory 1318, the domain experts 1320, and the domain tables

10 1324 operate in a manner similar to the manner described for similarly named items in Fig. 2 above. The rules provided in the SDL of discourse model 1316 also provide for generating scores for the various discourse semantics and for selecting a particular

15 discourse semantic to provide to a rendering engine 1326.

Continuing the example of the surface semantic described above, the output of the discourse engine would be an SML page containing:

```
20  <DirectoryQuery…>
     <Person id="kuansanw" parse="kuansan" score="99">
      <First>Kuansan</First>
      <Last>Wang</Last>
       ...
25   </Person>
     <DirectoryItem parse="phone number" score="45">
      <phone>+1(425)703-8377</phone>
     </DirectoryItem>
     </DirectoryQuery>
30
```

In this SML page, it can be seen that the discourse engine has resolved the phone number to a number entity and has resolved the reference to Kuansan to a particular person Kuansan Wang.

35 The SML pages provided by discourse engine 1314 are also decorated with inference scores as shown

above. Although the examples above only demonstrate well-behaved recognition and understanding, those skilled in the art will appreciate the XML-compliant nature of SML possesses sufficient expressive power in
5   annotating recognition and semantic ambiguities as they arise.

The SML pages generated by discourse engine 1314 are provided to rendering engine 1326. In some embodiments, the two engines are located on different
10 computers and are connected by a network such as network 1352 of FIG. 13.

At step 1404 of FIG. 14, rendering engine 1326 converts the SML pages it receives into an appropriate action. Initially, the XML pages received
15 by rendering engine 1326 are applied to a behavior model 1328. Under one embodiment of the present invention, behavior model 1328 is designed using an extensible stylesheet language (XSL) and in particular is designed using XSL-transformations (XSLT). Using
20 the XSLT standard, the behavior model can transform the SML structures into another mark-up language, say for example hypertext mark-up language (html), wireless mark-up language (wml) or a text-to-speech (tts) mark-up language. Thus, behavioral model 1328
25 includes rules for converting specific SML structures produced by the discourse engine into actions that are embedded within an appropriate output page such as an html page, a wml page, or some other output.

In other embodiments, the behavior model is
30 only able to use the document matching portion of the XSLT standard and not the document generation portion. This occurs when the action to be taken does not involve the production of a markup language page, for

example, when the action is a system command. In such cases, the behavior model performs the matching function and then determines the best means to generate the appropriate action.

5   Thus, the output of the behavior model can be action pages that include things like scripts that ask the user clarifying questions, or system commands that take specific actions.   Thus, rendering engine 1326 selects an appropriate action in a manner similar

10 to the way in which the rendering engine of Fig. 2 selects an appropriate action.

An example of an XSLT section appropriate for producing text-to-speech actions of the SML text described above is:

```
15  <xsl:template match="DirectoryQuery[@not(status)]">
        For <xsl:apply-templates select="Person"/>, the
        <xsl:apply-templates select="DirectoryItem"/>
    </xsl:template>
    <xsl:template match="Person">
20      <xsl:value-of select="First"/>
        <xsl:value-of select="Last"/>
    </xsl:template>
    <xsl:template match="DirectoryItem">
        <xsl:apply-templates/>
25  </xsl:template>
    <xsl:template match="phone">
        phone number is <xsl:value-of/>
    <xsl:template>
```

30 This XSLT text leads to a response provided to the user "For Kuansan Wang, the phone number is +1(425)703-8377." which would be provided as an audio signal to the user. Those skilled in the art will appreciate that advanced text to speech mark-up, such

35 as prosodic manipulation, can be easily added to the above example. The behavior model could alternatively select an XSLT stylesheet to render the response as an

html table. An example of a stylesheet that would produce such a table is shown below:

```
   <xsl:template match="DirectoryQuery[@not(status)]">
5      <TABLE border="1">
         <THEAD><TR>
          <TH>Properties</TH>
          <TH><xsl:apply-templates select="Person"/></TH>
         </TR></THEAD>
10       <TBODY><xsl:apply-templates select=
         "DirectoryItem"/>
         </TBODY>
       </TABLE>
   </xsl:template>
15 <xsl:template match="phone">
       <TR><TD>phone</TD><TD><xsl:value-of /></TD></TR>
   </xsl:template>
```

Note that the rendering engine can select the appropriate stylesheet template dynamically based on interface information in interface memory 1322 and the tags in the SML document describing the discourse semantic structure.

The SML discourse page provided by the discourse engine can include cues to help rendering engine 1326 determine what action to take. For example, if a query is for a person named "Derek" and there are 27 matches in the database, the discourse SML page looks like:

```
   <DirectoryQuery status="TBD" focus="Person" ...>
    <PersonByName type="Person" parse="Derek" status=
    "TBD" ...>
       <error scode="1" count="27"/>
35     <Person id="derekba">
         <First>Derek</First>
         <Last>Bevan</Last>
           ...
       </Person>
40     <Person id= "dbevan">
         <First>Derek</First>
```

```
        <Last>Bevan</Last>
          . . .
       </Person>
          . . .
5   </PersonByName>
       . . .
    <DirectoryQuery>
```

In this example, semantic objects that could not be collapsed by the discourse engine such as DirectoryQuery and PersonByName in the above example, are flagged with a status of to-be-determined "TBD". The discourse SML also marks the current focus of the dialog to indicate the places where the semantic evaluation is continued. For example, in the DirectoryQuery tag, the focus attribute is set equal to person indicating that the person associated with the directory query has not been resolved yet. These two cues assist the behavioral model in choosing an appropriate response i.e. an appropriate XSLT stylesheet.

In this example, the behavior model could select an XSLT stylesheet that produces an html page to present all of the 27 possibilities on a display. However, this would only be appropriate if the user had a full-scale browser available to them. If such a browser were not available to the user, the system could alternatively use a text-based stylesheet. However, such a stylesheet may require a more elaborate dialog strategy that would be based on several dialog turns in which the user is asked a sequence of questions to resolve the ambiguity.

The action determined by behavior model 1328 is implemented by rendering engine 1326, often resulting in an output to a user output interface

1330. This output may be passed directly to the user output interface or may pass through an intermediate network 1332.

5   One aspect of this embodiment of the invention is that the user is able to switch their user interface in the middle of a dialog session. To do this, the user communicates that they wish to switch to a new user interface. Discourse engine 1314 passes this interface information to rendering engine
10   1326 as the latest discourse semantic SML page. The behavioral model converts this SML page into an action that updates interface memory 1322 to reflect the newly selected interface. This new interface is then used by rendering engine 1326 to select the proper
15   stylesheets for future SML pages so that the SML pages are converted into an output format that is appropriate for the new interface.

Note that under this system, the discourse semantic structure for the discourse itself does not
20   change. As such, discourse engine 1314 does not need to be recoded or have its operation changed in any manner when the user changes the output interface. This makes it easier to adapt to new user output interfaces as they become available.

25   Since the dialog strategy is encoded in the XSLT that can be dynamically swapped, the system also exhibits tremendous amount of flexibility for dialog designers to dynamically adapt dialog strategies. For example, when the system encounters an experienced
30   user, the behavior model can choose to apply a stylesheet that lets the user to decide dialog flow most of the time. When confusions arise, however, the behavior model can decide to roll back to a "system-

initiative" dialog style in which the human user is asked to comply with more rigid steps. Changing dialog styles in the middle of dialog sessions amount to applying different stylesheets and requires no changes in the language or discourse models for the rest of the system.

One reason that XSLT has not been applied to dialog applications is that XSLT has no memory state. As such, the behavior model and the rendering engine in the embodiment of Fig. 13 are unable to store the past state of the dialog. Under the embodiment of Fig. 13, this does not represent a problem since discourse engine 1314 can manage and store the past states of the dialog. Discourse engine 1314 then passes any memory elements that are needed by rendering engine 1326 and behavioral model 1328 through the discourse semantic structure of the SML page.

Although the present invention has been described with reference to preferred embodiments, workers skilled in the art will recognize that changes may be made in form and detail without departing from the spirit and scope of the invention. In particular, although the invention has been described above with reference to tree structures, any suitable data structure may be used and the invention is not limited to a tree-based structure.

In addition, although the embodiments described above utilize a discourse semantic engine and discourse model, in other embodiments, these elements are not included. In such embodiments, the surface semantics are provided directly to the rendering engine, which then selects an action by

applying the surface semantics directly to the behavior model. In such embodiments, the behavior model determines the costs of the actions based on the surface semantic alone, without reference to a dialog state.